Linux 学习

目录

Linux 介绍

什么是 Linux

Linux 是 Unix 操作系统的继承和发展，是一个支持多用户，多进程，多线程，实时性较好，功能强大而稳定，同时具有良好的兼容性和可移植性，完全开放源代码的操作系统。open, free, share

Linux 是一个可以自由分发的，类似于 UNIX 的操作系统，它包括内核、系统工具、应用程序，以及完整的开发环境。

Linux is a free Unix-type operating system originally created by Linus Torvalds with the assistance of developers around the world. Developed under the GNU General Public License , the source code for Linux is freely available to everyone

Linux 是支持硬件平台最多的操作系统。

Linux 的五大支柱

我们可以对上述 Linux 的五大支柱归纳如下：

- UNIX 操作系统 -- UNIX 于 1969 年诞生在 Bell 实验室。Linux 就是 UNIX 的一种克隆系统。UNIX 的重要性就不用多说了。
- MINIX 操作系统 -- MINIX 操作系统也是 UNIX 的一种克隆系统，它于 1987 年由著名计算机教授 Andrew S. Tanenbaum 开发完成。由于 MINIX 系统的出现并且提供源代码(只能免费用于大学内)在全世界的大学中刮起了学习 UNIX 系统旋风。Linux 刚开始就是参照 MINIX 系统于 1991 年才开始开发。

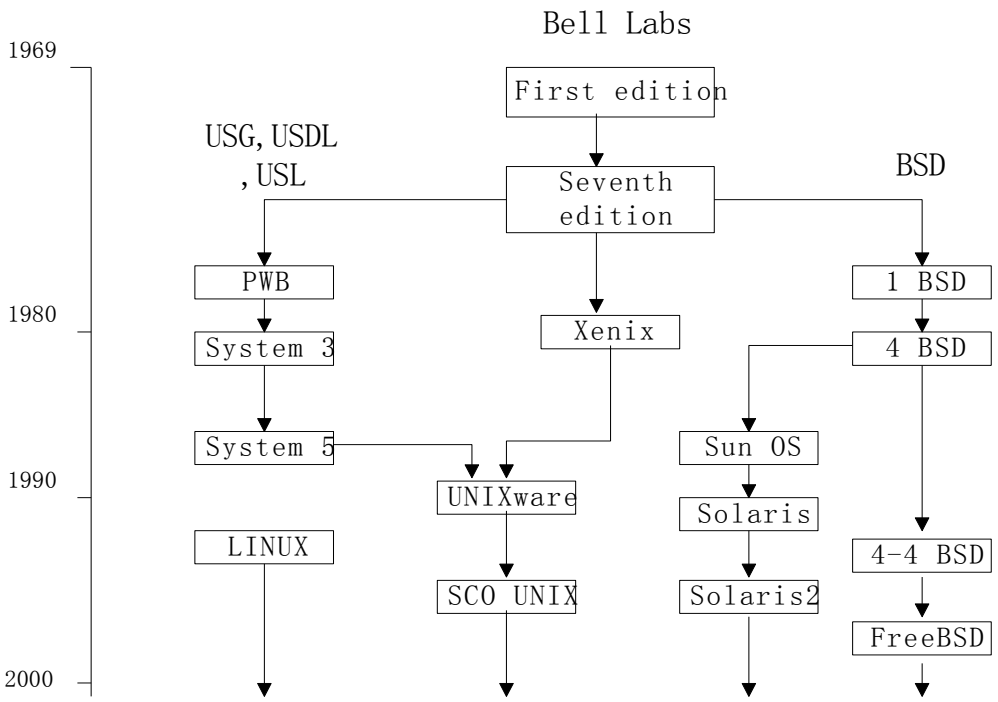
- GNU 计划-- 开发 Linux 操作系统，以及 Linux 上所用大多数软件基本上都出自 GNU 计划。Linux 只是操作系统的一个内核，没有 GNU 软件环境(比如说 bash shell)，则 Linux 将寸步难行。

- POSIX 标准 -- 该标准在推动 Linux 操作系统以后朝着正规路上发展起着重要的作用。是 Linux 前进的灯塔。

- INTERNET -- 如果没有 Intenet 网，没有遍布全世界的无数计算机骇客的无私奉献，那么 Linux 最多只能发展到 0.13(0.95)版的水平。

UNIX

Linux 的源头要追溯到最早的 UNIX。Linux 内核继承 UNIX 两大分支（UNIX system V 和 BSD UNIX）的主要特征，并进一步增加了新的特征。UNIX 是一个简单却非常优秀的操作系统模型。



• 60--70 年代　完成内核雏形

由 Bell 实验室的 K.thompson 、 Dennis Ritchic 在 PDP-7 上完成“太空旅行计划”而开始；(文件管理系统)

• 1978 年　UNIX 第七版问世

具有一定规模、功能完备，核心部分由 10000 指令构成，其中 90%是用 C 写的。

- 80 年代初　UNIX 商业化阶段

出现了 BSD 、Xenix 、PWB，核心是 UNIX 第七版，但加入了现代操作系统的特征；

- 1988 年后　UNIX 标准化阶段

AT&T 与 Sun：UNIX system V.4; IBM 与 HP:UNIX 标准版

- 90 年代后　并行处理及分布式网络系统

实现并行处理及针对可变动性强、对称多处理机的特征实现分布式处理，出现了各种硬件环境的 UNIX 系统，如：AIX、HP-UNIX、Solaris、 Linux

GNU

Richard Stallman 于 1984 年创立了 GNU 项目。该项目的目标是全部用自由软件建立完整的、自由的 UNIX 类操作系统。GNU 宣言（Manifesto）事先宣布了开发自由的 UNIX 类操作系统的目标，并把它称为 GNU 操作系统。GNU 的名称来自 GNU’s Not UNIX 的双重缩写。

Richard Stallman 在其他人的协作下创作了通用公共许可证（General Public License，GPL）。GPL 保证任何人有共享和修改自由软件的自由，任何人有权取得、修改和重新发布自由软件的源代码，并且规定在不增加附加费用的条件下得到源代码。

GNU 操作系统或 GNU 系统现在使用 Linux 内核，并且得到了广泛的应用。这一系统通常被称为 Linux 系统。

Linux 历史

1991 年，Linus Torvalds 是芬兰赫尔辛基大学的一名计算机科学系的学生 。

… I’m working on a free version of a Minix-lookalike for AT-386 computers. It has finally reached the stage where it’s even usable(though may not be depending on what you want),and I am willing to put out the sources for wider distribution.

年份	使用者数量	版　本[2]	大小（LOC）[1]
1991	1	0.01	10k
1992	1k	0.96	40k
1993	20k	0.99	100k
1994	100k	1.0	170k
1995	500k	1.2	250k
1996	1.5M	2.0	400k
1997	3.5M	2.1	800k
1998	7.5M	2.1.110	1.5M
2001		2.40	
2003		2.60	
2007.10		2.6.23	

- Linux 内核的版本号由 x.y.z 格式表示。x、y 和 z 是阿拉伯数字，诸如：2.0.36、2.2.7 等。
  - x：主版本号，只有在内核有重大改变的情况下，数值才会增加；
  - y：如果是奇数，表示该版本的 Linux 内核尚处于开发测试阶段；如果是偶数，则表示开发过程已经告一段落，系统已经相当稳定；
  - z: 辅助版本号，只要内核的代码一有变化，达到相对稳定，z 的数值就会增加。
- Linux 发行套件（Distribution）：以 Linux Kernel 为核心，搭配各种应用程序和工具。目前有 200 余种,常见的 Linux Distribution

西文版/国际版	中文版
RedHat Linux	Xteam Linux
Mandrake Linux	红旗Linux
Debian GNU/Linux	Turbo Linux 中文版
Slackware Linux	BluePoint Linux
SuSE Linux	
Turbo Linux	

Linux 性能

1.与 UNIX 操作系统兼容

- Linux 是按照 POSIX（Portable Operating System Interface for UNIX）1003.1 标准开发的操作系统;
- 它与 UNIX 在功能上完全兼容。

2.简单廉价的运行条件

- Linux 内核短小精悍，对运行条件要求十分简单。具有很好的稳定性和扩展性；
- 可以在廉价的 IBM PC 兼容机系列 386,486, Pentium 上运行。
- 甚至在只有 4MB 内存的 80386 机器上都可以正常使用。

3. 强大的网络功能

- Linux 内核支持各种网络协议，Ethernet，PPP、SLIP、NFS、AX.25、IPX/SPX、P(Novell)
- 通过 PPP，SLIP 可以使用 TCP/IP 网络服务等。
- 使用高速 Modem 通过电话线就可以接入 Internet 互联网。

Linux 特点

1. 多任务多用户

- 多任务：计算机可以在同一时间内运行多个进程，它们互不干扰、相互独立。
- 多用户：多个用户可以在同一时间内使用一台计算机。
- 各个用户通过终端，共享主机的资源，每一个用户都感觉到是自己在使用一台独立的计算机。

2. 多平台

- Linux 是在 80386 机器上开发的，所以它主要在 x86 平台上运行。
- 目前 Linux 已经移植到其它平台上运行，如 Alpha、Sparc、Mips、Amiga、PPC 等。

3. 设备独立性

- 为了在增加的新设备时不涉及内核，把设备看出一个独立的文件。
- 由内核对文件和设备提供统一的接口。
- 在增加设备时，只需把设备的驱动程序连接到系统中，系统就可以通过接口控制使用设备。

4．支持多种文件系统

- 由于 Linux 采用了虚拟文件系统 VFS，可以支持多种不同的物理文件系统。
- 包括 ext、ext2、minix、xiafs、hpfs、fat、msdos、umsdos、vfat、proc、nfs、iso9660、smbfs、ncpfs、affs、ufs、romfs、sysv、xenix、cohernet 等。
- 这些文件系统的文件可以装载到系统中，系统可以对这些文件进行访问和处理。

5．完善的虚拟存储技术

- 采用请求页式存储管理和交换技术。
- 为用户提供比实际内存大的多的虚拟存储空间。
- 每个用户可以使用 4GB 的虚拟内存空间。
- 各个用户的存储区域相互独立、彼此隔离。保证了系统和每一个用户的信息安全。

6．支持多种硬件设备

- Linux 支持的硬件设备种类相当广泛
- 硬盘驱动器、软盘驱动器、光驱、鼠标、键盘、扫描仪、打印机、数字化仪、声卡、Modem、到 MP3 播放器、手写板、游戏杆、USB 并/串口、码相机、
- 摄像头、USB HUB、SCSI 卡、声卡、磁带机、光驱、光盘刻录机、网卡、ZIP/MO 驱动器等。

Linux 与 Windows

	Linux	Windows
字符[3]	bash,ash,csh,netsh	cmd
桌面[3]	GNOME，KDE两种模式	单一模式
服务	DHCP,DNS,web,FTP,Samba, Proxy,mail,SSH,VNC,telnet （多提供了代理服务，mail服务）	DHCP,DNS,WEB,FTP,TELNET, 网络共享，超级终端
应用	OA，管理，多媒体，游戏	OA，管理，多媒体，游戏
网络	LAMP,VPN，NAT	VPN，NAT
安全	nmap, snort	X-Scan,木马冰河，海洋顶端

Linux 为建设网站提供较好的组合：

LAMP = Linux + arparch + mySQL+ PHP

- 字符用户界面/命令行用户界面 （CUI）计算机通过键盘等输入设备用于接收用户命令。如果命令只是通过键盘传递给操作系统，那么这一操作系统采用的是字符用户界面（character user interface），即通常所说的命令行用户界面（command line user interface，CUI）。如：DOS、Linux、UNIX。
- 图形用户界面（GUI），如果操作系统的最主要输入设备是鼠标等点击设备（point-and-click device），那么它采用的是图形用户界面（graph user line）。如：MacOS、OS/2、微软 Windows。
- Linux 的基本界面是字符界面，但它可以运行基于 X Window 系统（MIT 的 Athena 项目）的软件，X Window 提供 GUI 界面。
- 虽然所有 Linux 命令都是在字符界面下执行，但是许多因特网工具和软件开发工具都有图形界面。因此大多数 Linux 系统都会附带一个基于 X 的桌面环境的图形界面软件包。现在使用最广泛的桌面环境是：
  - ✓ GNOME： GNU 网络对象模型环境（GNU Network Object Model Environment）。GNOME 是 Red Hat Linux6.1 以及更高级版本的系统默认桌面环境。
  - ✓ KDE：K 桌面环境（K Desktop Environment，KDE）

Linux 的应用领域

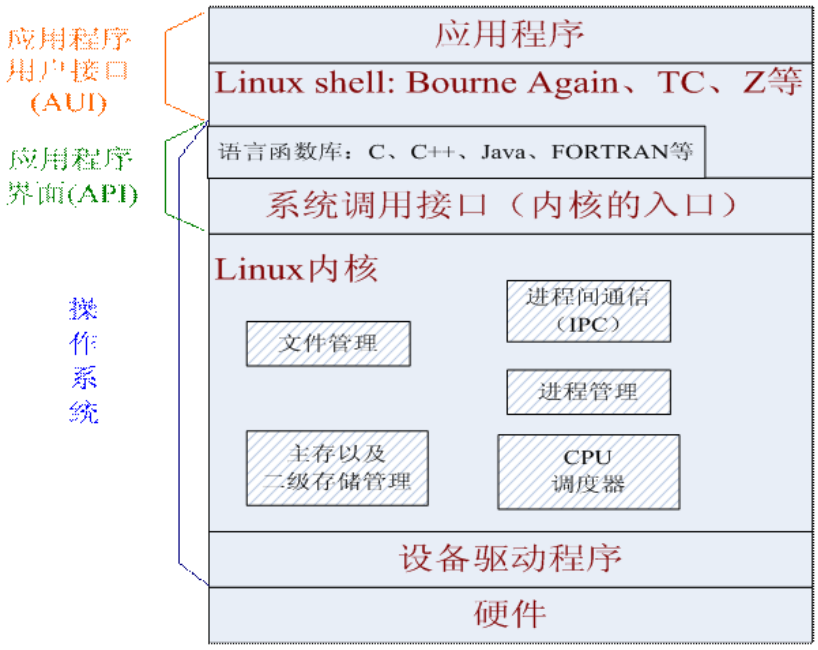
- Linux 服务器

- 嵌入式 Linux 系统
- 桌面市场

典型应用

- Titanic：特效 -> 350 台 SGI 及 160 台 DEC Alpha 工作站，有 105 台运行 Red Hat 4.1
- Linux 超级电脑：Los Alamos National Laboratory 利用 70 台 DEC Alpha 533Mhz，128M 内存，3G 硬盘的计算机，运行 Red Hat 5.0，造价 15 万美元，运算速度类似 SGI Origin 2000（造价 180 万美元）

Linux 软件体系结构



设备驱动程序层

设备驱动程序控制了操作系统和它所控制的硬件设备之间的交互。当用户命令或应用程序需要执行硬件相关的操作，例如文件读（转换成一个或更多硬盘读）时，这些程序代表 Linux 内核执行。用户没有直接访问这些程序的权限，所以不能把它们当成命令执行。

Linux 内核

Linux 内核包含了真正的操作系统。

Linux 内核层的主要功能：进程管理、文件管理、内存管理、磁盘管理

进程管理

内核管理进程的创建、挂起和终止，维护它们的状态。它也提供各种机制用于进程间相互通信，并且在分时系统中调度 CPU 以同时运行多个进程。

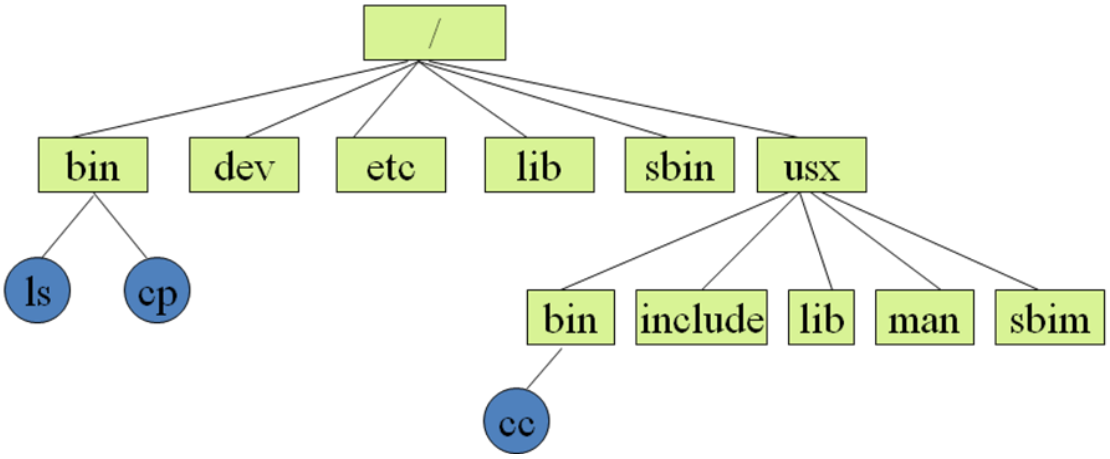
Linux 系统提供多种进程间通信(IPC)机制：

- 管道： 是可以运行在同一台计算机上的两个或更多相关进程用作 IPC 渠道。
- 命名管道（FIFO）：是驻留在磁盘上的永久通信渠道，而且可以被两个或者更多个运行在同一台计算机上的相关或不相关的进程用作 IPC。
- BSD 套接字：是允许在网络上（或在因特网上）的两个或者更多进程通信的临时渠道，虽然它可以被同一台计算机上的进程使用。

文件管理

内核管理文件和目录（文件夹）： 文件的创建和删除、目录的创建和删除，以及文件和目录属性的维护。

Linux 的一个重要特征是它支持多种不同的文件系统。



在 Linux 中，每一个硬件设备都映射到一个系统的文件，对于硬盘、光驱等 IDE 或 SCSI 设备也不例外。

Linux 把各种 IDE(Integrated Drive Electronic) 集成驱动器电子设备分配了一个由 hd 前缀组成的文件；而对于各种 SCSI (Small Computer System Interface) 小型计算机系统接口设备，则分配了一个由 sd 前缀组成的文件。

例如，第一个 IDE 设备，Linux 就定义为 hda；第二个 IDE 设备就定义为 hdb；下面以此类推。而 SCSI 设备就应该是 sda、sdb、sdc 等。

硬盘/光驱对照表

配置名称	说　　明
/dev/hda /dev/hdb	IDE I的Master/Slave硬盘/光盘
/dev/hdc /dev/hdd	IDE II的Master/Slave硬盘/光盘
/dev/sda /dev/sdb	第一，第二个SCSI硬盘
/dev/scd0 /dev/scd1	第一，第二个SCSI光驱

内存管理

内存管理以一种有序的方式分配和回收 RAM，以便每个进程得到足够的空间来正确运行。它确保分配给一个进程的部分或者全部内存空间不属于另一个进程。当进程终止时，内核回收分配给这个空间，并且这些空间放回空闲空间池以便重新使用。

Linux 采用了灵活的磁盘缓冲调度，能充分利用系统内空余的内存来提高 I/O 速度，又不妨碍规模大的应用程序运行。Linux 采用的 ext2 文件系统效率很高，而且采用了有效的机制防止文件碎片过度产生，对掉电或硬件损坏等原因造成的文件系统故障又足够的预防和恢复机制。另外，动态链接库技术、内存共享等技术的采用也提高了内存使用的效率。

Linux 还支持多达 2GB 的虚拟内存。

磁盘管理

内核负责维护空闲以及正被使用的磁盘空间，负责有序、公平地分配和回收磁盘空间。它决定在哪里，将多大的空间分配给一个新创建的文件。

内核负责磁盘调度，当同一个磁盘的多个请求（文件读、写等）到达时，决定下一次为哪个请求服务。

其它管理

内核完成一些其他工作来保证公平、有序和安全地使用计算机系统。这些工作包括管理 CPU、打印机和其他 I/O 设备。

内核确保没有用户进程永远占用 CPU，多个文件不会同时在一个打印机上打印，一个用户不能终止另外一个用户的进程。

系统调用接口

系统调用接口层包含进入内核代码的切入点。由于所有系统资源被内核管理，任何涉及访问系统资源的用户请求或应用程序请求，必须由内核代码处理。但是出于安全的原因，用户进程不能随意访问内核代码。

为了让用户进程能调用（启用）内核代码的运行，Linux 提供了一些方法或函数调用，称为系统调用。

系统调用允许用户操纵进程、文件和其他系统资源。

语言函数库

函数库时一组已经预先写好和测试过的可以被程序员用于开发软件的函数。这一层包含了几种语言的函数库，例如：C、C＋＋、Java 和 FORTRAN。

函数库和系统调用层形成常说的应用程序界面（API）。

Linux shell

Shell 是系统的用户界面，提供了用户与内核进行交互操作的一种接口(命令解释器) .

Linux shell 是从登录就开始运行并且解释输入的命令的程序。

它接收用户输入的命令并把它送入内核去执行

目前主要有下列版本的 Shell 有：

Bourne Shell：是贝尔实验室开发的。

BASH：是 GNU 的 Bourne Again Shell，是 GNU 操作系统上默认的 shell。

Korn Shell：是对 Bourne SHell 的发展，在大部分内容上与 Bourne Shell 兼容。

C Shell：是 SUN 公司 Shell 的 BSD 版本。

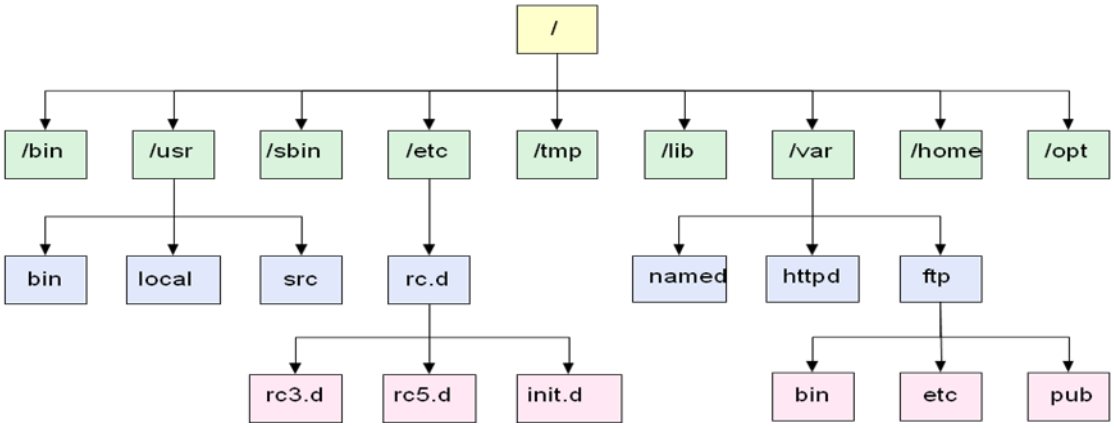
应用程序

应用程序层包括可以获得的所有的应用程序（工具、命令和实用程序）。当使用的应用程序需要操纵系统资源时，它需要调用完成相应任务的内核代码。

应用程序通过两种方法实现相应内核代码的执行：合适的库函数，系统调用。

库函数调用最终时用系统调用来启动合适的内核代码执行的。

linux 目录结构



linux 文件系统的入口，也是处于最高一级的目录

1. /bin　基础系统所需要的那些命令位于此目录，也是最小系统所需要的命令；比如 ls、mkdir 等命令；功能和/usr/bin 类似，这个目录中的文件都是可执行的，普通用户都可以使用的命令。做为基础系统所需要的最基础的命令就是放在这里。
2. /boot　linux 的内核及引导系统程序所需要的文件。
3. /dev 设备文件存储目录，比如声卡、磁盘、光驱等。
4. /etc 系统配置文件的所在地，一些服务器的配置文件也在这里；比如用户账户及其密码配置文件；
5. /home 普通用户家目录默认存放目录；
6. /lib 库文件存放目录
7. /lost+ found 在 ext2 或 ext3 文件系统中，当系统意外崩溃或机器意外关机，而产生一些文件碎片的文件系统，有时候系统发生问题，有很多的文件被移动到这个目录中，可能会用手工的方式来修复，或移到文件原来的位置上。
8. /mnt 这个目录一般是用于存放挂载储存设备的挂载目录的，比如有 cdrom 等目录。
9. /opt 表示的是可选择的意思，有些软件包也会被安装在这里，也就是自定义软件包，比如在 Fedora Core 5.0 中，OpenOffice 就是安装在这里。有些我们自己编译的软件包，就可以安装在这个目录中；
10. /proc 操作系统运行时，进程(正在运行中的程序)信息及内核信息(比如 cpu、硬盘分区、内存信息等)存放在这里。/proc 目录伪装的文件系统 proc 的挂载目录，proc 并不是真正的文件系统。
11. /root　linux 超级权限用户 root 的家目录;

12. /usr 这个是系统存放程序的目录，比如命令 、帮助文件等。

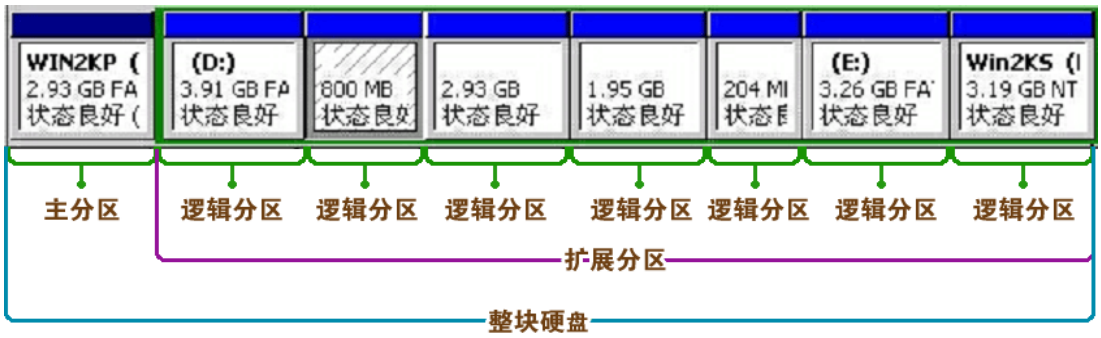
13. /var 这个目录的内容是经常变动的，看名字就知道，我们可以理解为 vary 的缩写，/var 下有/var/log 这是用来存放系统日志的目录。/var/www 目录是定义 Apache 服务器站点存放目录；/var/lib 用来存放一些库文件，比如 MySQL 库文件

14. /sbin 大多是涉及系统管理的命令的存放，是超级权限用户 root 的可执行命令存放地，普通用户无执行权限这个目录下的命令，这个目录和/usr/sbin; /usr/X11R6/sbin 或 /usr/local/sbin 目录是相似的；我们记住就行了，凡是目录 sbin 中包含的都是 root 权限才能执行的。

15. /tmp 临时文件目录，有时用户运行程序的时候，会产生临时文件。/tmp 就用来存放临时文件的。/var/tmp 目录和这个目录相似。

Linux 安装

磁盘及分区



有三种分区类型：主分区、逻辑分区和扩展分区。

分区表在硬盘的主引导记录（master boot record，MBR）上。MBR 是硬盘上的第一个扇区，所以分区表不能在其中占据太大空间。这将一个硬盘上的主分区数限制为 4 个。如果需要超过 4 个分区（这种情况很常见），那么主分区之一必须变成扩展分区。一个硬盘只能包含一个扩展分区。

扩展分区 只是逻辑分区的容器。MS DOS 和 PC DOS 原来使用这种分区方案，这种方案允许 DOS、Windows 或 Linux 系统使用 PC 硬盘。

Linux 的分区规定

设备管理

在 Linux 中，每一个硬件设备都映射到一个系统的文件，对于硬盘、光驱等 IDE 或 SCSI 设备也不例外。

Linux 把各种 IDE 设备分配了一个由 hd 前缀组成的文件；而对于各种 SCSI 设备，则分配了一个由 sd 前缀组成的文件。

例如，第一个 IDE 设备，Linux 就定义为 hda；第二个 IDE 设备就定义为 hdb；下面以此类推。而 SCSI 设备就应该是 sda、sdb、sdc 等。

分区数量

要进行分区就必须针对每一个硬件设备进行操作，这就有可能是一块 IDE 硬盘或是一块 SCSI 硬盘。

对于每一个硬盘（IDE 或 SCSI）设备，Linux 分配了一个 1 到 16 的序列号码，这就代表了这块硬盘上面的分区号码。

例如，第一个 IDE 硬盘的第一个分区，在 Linux 下面映射的就是 hda1，第二个分区就称作是 hda2。对于 SCSI 硬盘则是 sda1、sdb1 等。

各分区的作用

在 Linux 中规定，每一个硬盘设备最多能有 4 个主分区（其中包含扩展分区）构成，任何一个扩展分区都要占用一个主分区号码，也就是在一个硬盘中，主分区和扩展分区一共最多是 4 个。

对于早期的 DOS 和 Windows（Windows 2000 以前的版本），系统只承认一个主分区，可以通过在扩展分区上增加逻辑盘符（逻辑分区）的方法，进一步地细化分区。

主分区的作用就是计算机用来进行启动 操作系统 的，因此每一个 操作系统 的启动，或者称作是引导程序，都应该存放在主分区上。这就是主分区和扩展分区及逻辑分区的最大区别。

Linux 下面每一个硬盘总共最多有 16 个分区。Linux 规定了主分区（或者扩展分区）占用 1 至 16 号码中的前 4 个号码。

以第一个 IDE 硬盘为例说明，主分区（或者扩展分区）占用了 hda1、hda2、hda3、hda4，而逻辑分区占用了 hda5 到 hda16 等 12 个号码。

对于逻辑分区，Linux 规定它们必须建立在扩展分区上（在 DOS 和 Windows 系统上也是如此规定），而不是主分区上。

因此，我们可以看到扩展分区能够提供更加灵活的分区模式，但不能用来作为 操作系统 的引导。 除去上面这些各种分区的差别，我们就可以简单地把它们一视同仁了。

分区指标

对于每一个 Linux 分区来讲，分区的大小和分区的类型是最主要的指标。容量的大小读者很容易理解，但是分区的类型就不是那么容易接受了。分区的类型规定了这个分区上面的文件系统的格式。

Linux 支持多种的文件系统格式，其中包含了我们熟悉的 FAT32、FAT16、NTFS、HP-UX，以及各种 Linux 特有的 Linux Native 和 Linux Swap 分区类型。

Linux 帐户管理

Linux 基本命令

命令执行方式

命令使用方法

一般命令格式：command [-options] [arguments]

多命令行：% pwd ; ls -l　　//用逗号隔开，按顺序执行两个不同的命令

多行命令：% cc　hello_world.c　-lxm　-lxt　-lx11　-lm\
　　-o hello_world　　//命令太长用换行符隔开

Linux 基础命令汇总

（1）帮助命令: man、help、learn

（2）登录命令: login, passwd

（3）用户命令: useradd, adduser

（4）常用命令: cat、more、less、head、tail、ls、cd、pwd、mkdir 、touch、rmfile、rm、cp、mv、chmod、chown、chgrp 、ln、file、find、grep、sort、who、whoami、id、passwd 、w、ps、kill、history、!!、wc、tab、date、cal、monut 、df、su、shutdown、poweroff、reboot、clear

（5）编辑器使用命令：

（6）网络命令：telnet、ftp、talk

（7）档案管理：rpm、tar、gzip/gunzip、zip/ unzip 等命令

find　搜索文件

find　/　-name　filename　//从根目录查询文件名为 filename 的文件

grep　查找匹配信息

格式:　grep 匹配内容 文件。

命令选项：　-c---显示匹配模式的行数

-i---匹配时忽略大小写

-l---仅显示有匹配模式的文件名

-n---显示匹配行的行号

-v---显示不匹配的行

例:　$ grep　-li　UNIX　text?　//查找 text 开头的文件下的 Unix 模式

输出: text1: UNIX

text2: Unixsystem

text3: unix

//查找当前目录下面所有文件里面含有 success 字符的文件

$ grep success　*

sort　对各行按词组或字符分类排序

选项：　-b---忽略文件中的空格字符

-d---字符按字典顺序排序

-f---按大写字符排序，将小写改成大写

-n---按数字的数值大小排序

-o---排序输出存入文件

-r---排序按字典反序进行

例：　$ sort　-fn　text1

$ sort　-f -o　text0　text1

Tab 键 自动补全名字

对某个文件或目录记得不全，只记得开头的字母，那么按 Tab 键自动补全名字。

passwd ：可以设置口令

history ：用户用过的命令 eg: history //可以显示用户过去使用的命令

!! ：执行最近一次的命令

su 切换用户 ：从 root 用户可以不需要密码切换到其他任何用户

Shutdown -now ：关机，只有 root 用户可以使用

Reboot ：重启

Poweroff ：直接关掉电源

clear ：清屏

w ：显示那些用户在线

11. su 在不退出登陆的情况下，切换到另外一个人的身份

用法: su -l 用户名(如果用户名缺省，则切换到 root 状态)

eg:su -l netseek (切换到 netseek 这个用户，将提示输入密码)

12.whoami，whereis，which，id

//whoami: 确认自己身份

//whereis: 查询命令所在目录以及帮助文档所在目录

//which: 查询该命令所在目录(类似 whereis)

//id: 打印出自己的 UID 以及 GID。(UID:用户身份唯一标识。GID:用户组身份唯一标识。每一个用户只能有一个唯一的 UID 和 GID)

eg: whoami //显示你自己登陆的用户名

whereis bin 显示 bin 所在的目录，将显示为：/usr/local/bin

which bin

20. finger 可以让使用者查询一些其他使用者的资料

eg: finger //查看所用用户的使用资料

finger root //查看 root 的资料

df 查看分区

压缩命令 tar

tar cvf filename.tar filename

-c 表示归档

-v 表示显示信息

-f 表示使用归档名称，必须是最后一个参数

-x 表示解除归档文件

gzip 压缩文件为 gz 的格式 gunzip 是解压缩命令

使用：gzip/gunzip 文件名

zip 将文件压缩为 zip 的格式 unzip 解压缩

使用： zip 包名 文件名 unzip 包名

挂载命令

monut 为 root 用户命令，加载一个硬件设备

用法:mount [参数] 要加载的设备 载入点

-a 加载文件/etc/fstab 中设置的所有设备。

-f 不实际加载设备。可与-v 等参数同时使用以查看 mount 的执行过程。

-F 需与-a 参数同时使用。所有在/etc/fstab 中设置的设备会被同时加载，可加快执行速度。

-h 显示在线帮助信息。

-L<标签> 加载文件系统标签为<标签>的设备。

-n 不将加载信息记录在/etc/mtab 文件中。

挂载光驱

mount /dev/cdrom /mnt/cdr //如果 cdr 目录不存在，先创建 cdr 目录

/dev/cdrom 为挂载的设备名称， /mnt/cdr 为设备挂载点(设备在那个目录下打开)

Umount /dev/cdrom (或/mnt/cdr)

挂载 u 盘

/fdisk –l 或 /sbin/fdisk –l 查看分区设备(如看到分区 sdb)

mount /dev/sdb /mnt –o iocharset=gb2312

VI 编辑器

vi(visual interpreter),是 UNIX 中基本编辑工具.

1. vi 的操作方式

包含三种方式:

1) 命令行方式

vi 的初始方式,完成:光标移动、删除字符、复制、写盘。

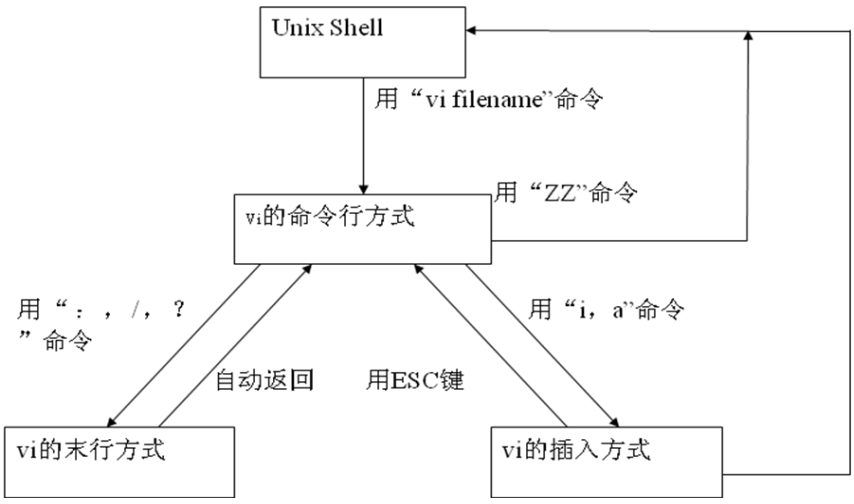
2) 插入编辑方式

可添加/输入文本及程序代码，非初始态，用 i/a 命令切换，[ESC]结束。

3) 末行命令方式

屏幕底部出现的命令行，由“：”、“/”、“？”进入，但是不同的命令进入只能进行相应命令下的操作。

2. vi 中多种工作方式的转换



3. 命令行方式的常用命令

注意命令的大小写，大写命令需加 shift 键。

操作符	操作效果
h(←)	光标左移一个字符
l(→)	光标右移一个字符
k(↑)	光标上移一行
j(↓)	光标下移一行
G	光标移至文件最后一行
0	光标移至首行
$	光标移至行尾
H	光标移至屏幕的最上行
M	光标移至屏幕的中部
L	光标移至屏幕的最下行
w	光标右移一个单词
ZZ	必要时写盘，并退出编辑

4. 退出编辑器命令项

按 shift + ： 时；表示编辑器处于等待用户输入控制命令的状态。

:q 无修改退出

:q! 不保存修改退出

:w 保存编辑内容

:w 文件名 保存另一文件中

:wq 保存修改退出

:w! 强制保存。

:w! file 强制保存到文件中。

:wq! 强制保存修改文件并退出

:set nu 显示行号

:set nonu 不显示行号

/exp 从光标处向前寻找字符串 exp

?exp 从光标处向后寻找字符串 exp

:n 重复前一搜索命令

:N　　在光标处向反方向重复前一搜索命令

5.vi 光标操作命令

b 移动到当前单词的开始

e 移动到当前单词的结束

w 向前移动一个单词

h 向前移动一个字符

j 向下移动一行

k 向上移动一行

l 向后移动一个字符

按两次 d 键删除光标当前所在行

按 u 键恢复原来操作

按两次 y 键复制光标当前所在行

按 p 键粘贴

按 c 键剪切

split 横向分屏　ctrl+w 连续按两次 在几个窗口间切换

vsplit 纵向分屏

Shell 的基本概念

Shell 是 UNIX 提供的与用户交互的接口，一般系统会提供多种 shell，如：Bourne shell，bash，c_shell，　korn shell 等。不同 shell 功能侧重点

- K-shell　B-shell 有程序设计优势
- c-shell 在命令使用中有优势，符合 C 编程习惯
- Bash、TC 和 Z shell 是新 shell 版本,功能多

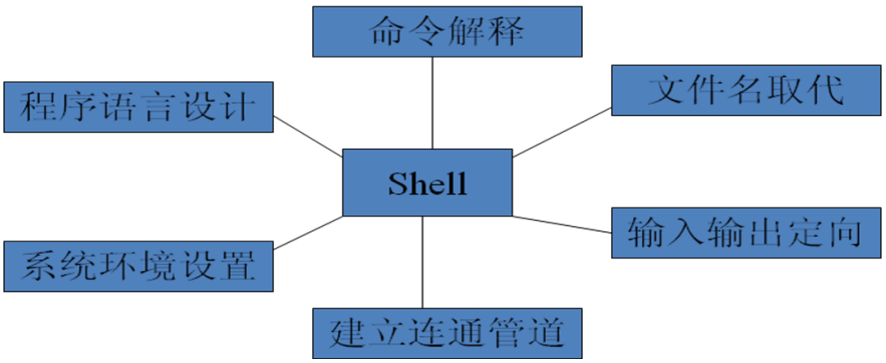
改变 shell 环境

在命令行输入 shell 程序名，输入 exit 返回原注册 shell(原注册 Shell 为 bash).

- # echo $shell //查看当前 shell
- # shell 名 //例如输入 csh 则进入 csh，输入 exit 则退出 csh 回到默认的 bash

shell 名	位置	程序（命令）名
Rc	/usr/bin/rc	rc （失败，没找到文件）
Bourne shell	/usr/bin/sh	sh
C shell	/usr/bin/csh	csh
Z shell	/usr/local/bin/zsh	zsh （失败，没找到文件）
Korn shell	/usr/bin/ksh	ksh
TC shell	/usr/bin/tcsh	tcsh

shell 基本工作内容



输入输出重定向

UNIX 系统标准流

- 数据流的概念： 将命令处理中的流看成“命令输入流”，“命令输出流”。
- Unix 标准流的概念：

文件描述	标准定义文件	实际对象
0	标准输入 stdin	键盘
1	标准输出 stdout	显示器
2	标准错误 stderr	错误信息（显示器）

输入输出重定向

用 > < >>等符号改变标准流的方向

- 输入重定向（<）,用来断开键盘和“命令”的标准输入之间的关联，将输入文件关联到标准输入。
- 输出重定向（>）,用来断开“命令”的标准输出文件和显示器之间的联系，并将输出文件和标准输出建立关联。

```
% cat  < tempfile

% ls -l  >dir1  //若文件不存在则自动创建

% pwd  >dir1

% date >>dir1  //追加
```

下面输入输出重定向结合使用:

```
% cat < infile >outfile  //相当于 cp 操作，只是 cp 的源文件一定要存在
```

错误流重定向

通常标准输出和错误输出流定向到显示器上，也可对其进行修改，不同的 shell 修改方法不同.

- C shell:
```
% cc abc.c >log  //编译 abc.c 文件并把正确信息输入到 log 文件。

% cc abc.c >& log  //编译 abc.c 文件并把错误信息输入到 log 文件。
```
- K-shell:
```
$ cc abc.c 2>&1 log //将正确的信息输入到 log 中

$ cc abc.c 2>  log  //将错误的信息输入到 log 中
```

管道线的处理

管道线命令体现了一种工作的思想——利用简单的编程方法和命令组合功能完成复杂的处理任务。管道线将命令隔开，同时将一个程序（或命令）的输出作为另一程序（命令）的输入。

例：
```
% ls –l > tempfile

% wc -l  tempfile  //用 wc 命令计算 tempfile 中的行数，wordcount

% rm  tempfile
```

用下面命令替换:

% ls -l | wc -l　　//管道线可看成是输入输出重定向的组合方式

管道线还可完成更为复杂的处理，如：

- 逻辑“或”：% write zhang <letter || mail zhang < letter

  //只有在前面执行错误的情况下才执行后面的命令

- 逻辑“与”：% mail zhang < letter　&&　rm　letter

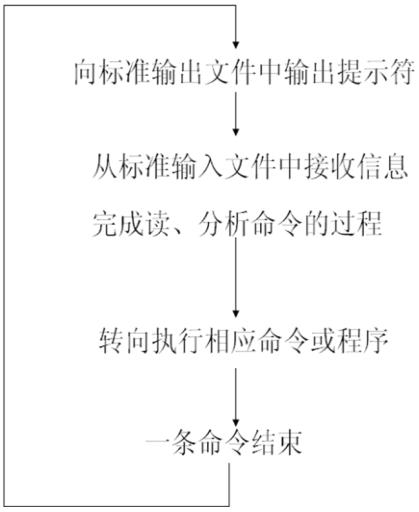
  //只有在前面命令执行正确的情况下才执行后面的命令

- 在 k sheel 及新版 shell 中有 tee 命令，完成三通管的作用：

  $ ps -ef　| tee　program.ps

命令解释

- Shell 是 Unix 的命令解释程序，其解释执行命令的过程如下：



- 命令解释中对特殊字符的解释：

  ;　命令行结束，但不换行

  &　后台命令　% cc　prgm.c　& //编译 prgm.c 的进程作为后台进程

  ( )　生成一子进程完成括号内的命令

  | ^　管道线标志（早期 Unix 使用^）

  < >　输入输出重定向

  $　shell 变量的取值与访问

! C shell 的历史纪录标志

\ 转义符，使紧跟其后的字符失去 shell 中的特殊含义

系统环境设置

不同的 shell 有不同的工作环境模式。 UNIX 启动时有环境设置文件支持，且 shell 不同运行的文件不同，它为用户建立独特的工作环境。shell 可提供维护环境变量服务：环境变量----是当前执行环境可修改的量。分为两类：

- 10--30 个环境变量与用户登录标识相关联，自动生成、由 shell 维护、随用户注销而消失；
- 用户私有环境变量,有特殊目、给用户很大的设置自由度。

- Shell 的变量及变量使用
  - 变量形式为：name = value 例：% SAMPLE="hello world"
  - 引用变量时： % echo $SAMPLE
  - 对已设置好的变量可使用 env 命令查看。
- 环境变量的保存文件
  - 完成 Shell 初始化文件是/etc/profile---设置全局变量
  - 保存局部变量的设置文件:
    - B-shell: .profile
    - K-shell: .kshrc /.profile
    - C-shell: .cshrc, .login

这些文件是隐含文件，可用编辑工具进行编辑。

在 korn shell 中与 .profile 执行有密切关系的还有两文件：

- .kshrc:存放其它环境变量，并由 .profile 提交执行
- .logout:存放退出 shell 时应执行的操作变量，也由.profile 提交执行。

- Korn shell 环境变量的设置， profile 实例

```
PATH=/usr/bin:/etc:/usr/sbin:/usr/lib:$HOME/bin:/sbin/bin
MAILCHECK=1
MAILPATH=/uer/spool/mail/user/
```

```
MAIL=/usr/spool/mail/user
MAILMSG="you have new mail/a"
export PATH MAIL MAILCHECK MAILPATH MAILMSG
export TMOUT=200
if test -s "$MAIL"
    then   echo "$MAILMSG"
fi
set -o ignoreeof
set -o noclobber
set -o vi
alias   dir="ls -l"
alias cls="clear"
```

- k-shell 环境变量定义

PATH—查询程序的路径

PS1—shell 的主提示符

PS2—shell 的第二提示符

MAILPATH—用户的电子邮箱路径

SHELL—使用 shell 程序的路径

PWD—当前工作目录

TMOUT—无命令输入退出等待时间（秒）

TERM—终端类型

HOME—用户主目录

EDITOR—缺省的编辑器

FCEDIT—fci 调用的编辑器

HISTFILE—历史命令文件名

- 开关选项变量（-o/+o)：

VI—使用内部编辑器

verbose—执行命令前显示

privileged—不运行用户的初始文件

ignoreeof—忽略 ctrl-d

noclobber—重定向不覆盖文件内容

allexport—对定义的变量自动传递

bgnice—以低优先级运行后台作业

- c shell 中环境变量的设置

在 c shell 中有两个文件共同完成环境变量的设置：

- .cshrc —用户注册后，每用 csh 创建一个进程时，系统执行一次此文件；
- .login —用户注册时执行一次，通常存放固定环境变量

- C shell 环境变量设置实例 .cshrc 文件：

```
set path=(.~/bin /usr/bin /usr/ucb /user2/ motif/bin)
set prompt=" 'whoami'@'hostname'\!)"
set filec
set history=50
set savehist=50
umask    027
alias cp cp-I
alias mv "mv -I"
alias rm `rm -I`
alias ls ls -sf
alias m more
alias h history
```

- C shell 的.login 文件：

```
Set    ignoreeof
setenv    more '-c'
setenv    PRINTER laser
setenv EXINT 'set sm ai nu ts=4'
stty -crterase
stty -tabs
stty crt
stty erase '^?'
stty werase '^h'
stty kill '^u'
```

shell 编程

Shell 程序的执行

shell 程序的功能是将命令序列（date　ls -l） 组合并由系统自动执行。file.sh 文件交给系统执行可用三种方式实现：

- 利用输入输出重定向　$ sh <file1.sh
- 把 file1.sh 当作 sh 的执行参数　$ sh file1.sh
- 直接执行 file1.sh 程序　$ file1.sh

Shell 变量的应用

Shell 变量是字符或字符串.

1) shell 变量及赋值

以字母开头,由字母、数字及下划线组成如：

ux=u.unix　　c shell 赋值：set　ux=u.unix

hi="how are you"　　set　hi="how are you"

2）shell 变量引用

在变量名前加"$",如：

echo $ux

echo ${ux}　或　echo　${ux}tm　也可用双引号做变量替换：

$echo "$ux" tm　或　echo "$hi" today

3）Shell 编程中三种引号的作用

- 单引号　' '：其中内容是字符串，没有 shell 的特殊含义；
- 双引号　" "：其中特殊字符作 shell 含义解释，其它作字符；
- 反引号 ``：其中的命令可作为执行结果进行赋值。

例：$ file=report

$ echo 'The time is `date`,the file is $file'

回显：　The time is `date`,the file is $file

$echo "The time is `date`,the file is $file"

$echo “The time is $(date),the file is $file”

回显：The time is wed Aug 16 15:11:42 Roc 2000,the file is report.

注：Red Hat 中利用$(date)

4) 变量的作用域：局部变量和全局变量

只有使用了 export 命令后，变量才可在子进程中起作用：

```
$ ux=unix
$ echo $ux    //显：unix
$ sh                  //进子进程
$ echo $ux    //显：
$ exit
$ echo $ux    //显：unix
$ export    ux
$ sh                  //进子进程
$ echo $ux    //显：unix
$ exit
```

5）环境变量的设置与取消

- 在 shell 初始化文件中设定环境变量，方法是：
  - K-shell 及 B-shell 中　　Variable_Name = Value
    export　Variable_Name
  - C-shell 中　　setenv Variable_Name = Value
- 删除环境变量
  - K-shell 及 B-shell 中：　　unset Variable_Name
  - C-shell 中:　　unsetenv Variable_Name

6）标准变量

Shell 标准变量，在 shell 进程创建时自动赋值：

变量用途	sh变量	csh变量	环境变量
用户名		user	USER
注册目录	HOME	home	HOME
访问路径	PATH	path	PATH
cd路径	CDPATH	cdpath	
提示符	PS1	prompt	
辅提示符	PS2		
终端类型	TERM	term	TERM
运行的shell	SHELL	shell	

7) 位置变量

Shell 中命令行的参数可用单独的位置参数提取,它们是$1,$2,$3,…$9. 考虑下列 shell 程序,其文件名为 echoarg.sh:

```
echo   $#
for   VAR   in   $*
   do   echo $VAR
done
```

执行此程序时用:

```
$ echoarg.sh   first   second   third
```

结果:

```
3
first
second
third
```

其它特殊的shell变量:

B shell	C shell	变量作用
$#	$#argv	位置参数个数
$?	$status	前命令返回状态
$$	$$	当前shell的pid
$!		最近访问的后台pid
$*	$*	用单字符串显示传递参数

8) 变量替换中的多值选择

程序中若有以下语句: echo “The CDPATH is $CDPATH” 会显示:The CDPATH is

对未赋值变量的赋值方法:

- ${var:-word}: 若 var 有值且非空，取该值，否则取 word, var 不变；
- ${var:=word}: 若 var 有值且非空，取该值，否则取 word,同时将 word 值赋给 var .
- ${var:+word}: 若 var 有值且非空，取 word,否则 var 仍为空.

可将上列语句改为：echo “The CDPATH is ${CDPATH:-Undefined}”

9) 用命令替换变量

指使用命令执行结果替换变量值，即用反引号完成命令替换。

3. Test 命令

格式：test expre 或 [test]

功能：当表达式的值为真时,命令返回真值 0,否则返回假值非 0。可完成对文件、字符串、数字、表达式进行判别及求值。

1）用于对文件的测试

test -[dfrmxs] file

其中：-d file 判文件存在且为目录文件

-f file 判文件存在且为普通文件

-r file 判文件存在且为可读文件

-w file 判文件存在且为可写文件

-x file 判文件存在且为可执行文件

-s file 判文件存在且长度为非零

例： test –d /home/usera && echo “目录 usera 存在”

test -d /home/usera || echo “ 目录 usera 不存在或无此目录”

2）对字符串 S 的测试

test s 字符串 S 为非空串时返回真值

test -zs 字符串为空时返回真值

test s1=s2 字符串 s1 与 s2 相同时为真值

3）对整数 n 进行测试

test n1 -eq n2 整数 n1 和 n2 相等时为真值

test n1 -ne n2 整数 n1 和 n2 不相等时为真值

test n1 -lt n2 整数 n1 小于 n2 时为真值

test n1 -le n2 整数 n1 小于等于 n2 时为真值

test n1 -gt n2 整数 n1 大于 n2 时为真值

test n1 -ge n2 整数 n1 大于等于 n2 时为真值

测试整数 n 时注意问题:

x1=“005”

x2=5

test “$x1”=“$x2”

echo $? //系统显示: 1 测试结果为假,两者不同,作为字符串不相同

test “$x1” -eq “$x2”

echo $? //系统显示: 0 测试结果为真,两者相同，作为数字值相同

4 .条件控制语句

在 shell 中用 if 和 case 语句形成程序的分支结构

1）if 语句

```
if    if_list
      then   then_list
elif    elif_list
      then    then_list
....
else    else_list
fi
```

将 test -d /home/usera && echo “目录..” 改为 if 语句实现：

```
if  test  -d  /home/usera
      then  echo “目录 usera 存在”
fi   //或写成：
if  [-d  /home/userd]
      then  echo “目录 usera 存在”
fi
```

例 1:使用 if 及 test 完成测试命令行参数的个数:

```
# !/bin/sh   filename: ifparam
if [$# -lt 3]; then
      echo   “usage: `basename $0` arg1 arg2 arg3”>&2
      exit 1
fi
echo “arg1:$1”
echo “arg2:$2”
echo “arg3:$3”
```

运行: $ ifparam cup medal

```
$ ifparam cup medal    trophy
```

例 2:将位置参数中的内容传递到系统命令中

```
# !/bin/sh    filename:testdir
# 将 $1 传递到命令的变量表中
DIREC = $1
if    [ `ls -a $DIREC` = “”];    //    if[-z    `ls -a $DIREC`]
   then
      echo “$DIREC is indeed empty”
 else
     echo “$DIREC is not empty”
fi
```

运行: $ testdir dirname

例 3:考虑一个实际应用问题：

设有一个连续运行系统，每当运行中遇到错误时，创建一个文件 errorfile 并将错误信息写入其中；要求我们编写一段 shell 程序生成错误日志文件，即写一个名为 checkerr 程序，每小时运行一次记录这些错误。

具体做法是：如果 errorfile 存在，checkerr 把日期、时间、错误信息一同记入 datelog 文件中，然后删除 errorfile ;若 errorfile 不存在，则记下日期、时间并给出无错误提示一并写入 datelog 中。程序编码:

```
#！ /bin/sh
# 例题  checkerr.sh
date >>datelog
if    test -r errorfile
    then    cat    errorfile >>datelog
                rm errorfile
    else      echo “No error this hour”>>datelog
```

```
fi
```

2）case 语句

语法：

```
case  word  in
        pattern -1) pat1 -list1;;
        pattern -2) pat2 -list2;;
        ……
                *) default -list;;
  esac
```

其中：word 将与各匹配模式比较，“;;”符表示匹配结束，“*”号为通配符。

例 1:向指定的文件中添加文本

```
#! /bin/sh
# filename:append.sh
case $#  in
        1)  cat >> $1;;
        2)  cat>>$2<$1;;
        *) echo  ‘usage: append.sh  [from] to ‘;;
 esac
```

执行：

```
$ append.sh  file1
$ append.sh  file1  file2
```

例 2：写一段根据一天中不同时间给出问候信息的程序

```
# !/bin/sh
# 练习 wh.sh  case 结构
hour= `date+%H`
case  $hour  in
    0[1-9]|1[01])  echo “Good morning!”;;
```

```
        1[234567])    echo “Good afternoon!”;;
        *)            echo “Good evening!”;;
    esac
```

例 3:用 case 语句测试命令行变量,完成不同的工作内容.

```
    # !//bin/sh    filename:case_param
    echo    “Enter your option and hit <Enter>:\c”
    read option
    case    “$option”    in
        d) date;;
        l) ls;;
        w) who;;
        q) exit 0;;
        *) echo “invalid option; try running the program again.”;    exit 1;;
    esac
    exit 0
```

5. 循环语句

循环语句完成命令的重复执行

1）for 循环

语法：for var in word1 word2 … wordn

```
        do  commands
    done
```

for 循环中可根据需要进行嵌套

例 1：列出用户注册目录下的 cc 和 work 子目录中所有 * .c 文件。

```
    # ！/ bin/sh
    # 显示.c 文件
```

```
cd    $HOME
for    dir in    cc    work
    do    echo “….in $dir….”
    cd    $dir
            for    file    in    *.[c]
                    do    ls -l    $file
            done
        cd
    done
```

2) while 循环

以命令表的出口状态为判别条件，决定循环体中的命令是否执行。语法结构:

```
while    cmdlist1
    do    cmdlist2
done
```

例 1：

```
#！/bin/sh
 # while.sh
 while [-r filea]
   do    echo ‘before sleep’
         sleep 5
         echo    ‘sleep done’
   done
```

例 2：

```
#! /bin/sh
 # cfile.sh
 VAL=1
 while [$VAL    -lt 11]
```

```
do    touch    file$VAL
        VAL=`expr $VAL+1`
done
```

例 2 中几个新内容：

- 命令 touch 的功能：改变文件访问权限和修改时间，用法：touch[选项] [时间] 文件名；若指出的文件名不存在时，则创建具有缺省权限及当前时间的文件。
- expr 的用法：将实参作为表达式求值的一种方法

```
 $ count=0
$ count=`expr $count+1`   (用 count =$count +1 不行)
$ echo $count
```

另外 while 循环中还可以使用：

- break---退出当前所在的整个循环
- continue---结束本轮循环，转到下一轮循环的开始

在 while 循环中的 cmdlist1 的设计要注意不能发生死循环.

6. 读取标准输入语句

将标准输入的值存入到变量中用 read 命令.

例：

```
# the read command example
echo “enter your name:\c”
read    name
echo    “your name is $name”
```

read 命令中的参数说明:

```
read [word1][word2][rest]    存放方式为:第一,第二,其余参数
```

例： read 命令参数使用

```
# test read command, filename:read_test
```

```
echo “give me a long sentence:\c”
read word1 word2 rest
echo “$word1\n $word2\n $rest”
echo “end of my act”
```

运行: $ read_test

give me a long sentence:

输入:let’s test the read command.

输出: let’s

test

the read command

end of my act

bash 的内部命令

echo eval exec export readonly read Shift wait exit

7. Shell 程序的调试

1）交互调试

先用命令验证，再引入 shell 编程中

2）在编辑过程中执行程序

打开多个窗口，边编辑边执行

3）用 shell 程序跟踪执行

使用 shell 的 -v,-x 选项可对 shell 程序进行跟踪

$ sh -v test.sh

$sh -x test.sh